TEC | Tecnológico de Costa Rica

Escuela de Computación

IC3002 Análisis de Algoritmos

Proyecto 2 de Análisis de Algoritmos

Profesora:

Lorena Valerio Solís

Integrantes:

Paola Melissa Alguera Castillo 2019056061

Richard Osvaldo León Chinchilla 2019003759

Adrián José Herrera Segura 2019081563

Fecha de entrega:

17 de Noviembre

San Carlos

2021

Introducción

Los juegos han evolucionado junto con las personas a lo largo de la historia, con el avance de la tecnología, la manera de crearlos, jugarlos y formular soluciones ha sido todo un reto para los humanos y el caso planteado en este proyecto no es la excepción.

El Tetravex es un juego de computadora donde se presentan un número x de piezas cuadradas las cuales, cada una cuenta con 4 números en cada extremo del cuadrado. En la versión de computadora, sobre la pantalla siempre habrá una cuadrícula donde se tendrán que colocar las piezas proporcionadas por el juego, las piezas no pueden ser colocadas de manera aleatoria, ni ser rotadas, la regla es que solo se pueden emparejar las piezas que posean el mismo número en la posición adyacente.

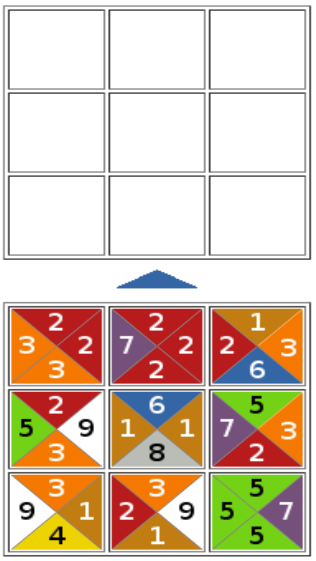


Figura 1: ejemplo tetravex

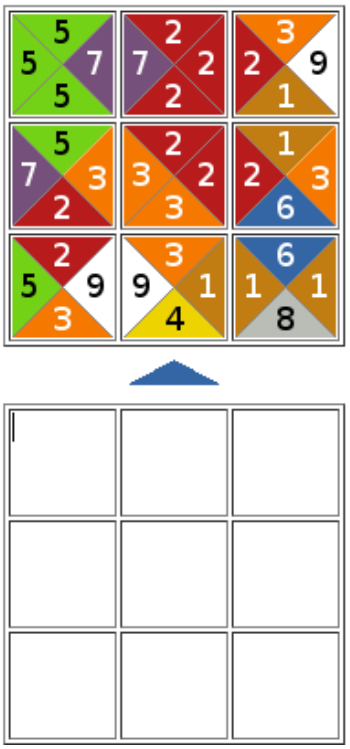


Figura 2: Tetravex armado.

El Tetravex fue inspirado por "el problema de embaldosar el avión" fue descrito por Donald Knuth en el volumen 1: Fundamental Algorithms. Fue nombrado por Scott Ferguson, el líder de desarrollo y arquitecto de la primera versión de Visual Basic. Algo importante de destacar es que este problema tiene una solución en general de NP-completo.

Para este segundo proyecto se incorpora un concepto nuevo de estrategia de algoritmos llamado: Algoritmos genéticos. Pero, ¿Qué es un algoritmo genético? Primeramente se debe dar un contexto referente a este concepto. En la teoría de evolución dada por Charles Darwin, padre de la teoría de la evolución, la selección es un proceso de supervivencia del más fuerte.

El investigador de la Universidad de Michigan, John Holland, a fines de los 60’s desarrolló una técnica que permitió incorporar a un programa, el objetivo era lograr que las computadoras aprendieran por sí mismas. Su algoritmo se llamó en un principio “planes reproductivos” pero se popularizó bajo el nombre de algoritmos genéticos.

Los algoritmos genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Estos están basados en los procesos genéticos de los organismos vivos, seleccionar a los más aptos para transferir sus características a la siguiente generación.(Valerio, 2021,diapositiva 5)

Estos algoritmos son una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia para la nueva generación.(Valerio, 2021,diapositiva 7)

En este proyecto se usarán principalmente los algoritmos genéticos para la resolución del tetravex basados en el principios de estos para lograr respuestas eficientes.

Análisis del problema

El proyecto consiste en realizar una medición y rendimiento de un algoritmo genético en el proceso de armar un rompecabezas tetravex. Para poder encontrar una solución, primeramente se deben generar el total de piezas según la dimensión, de forma aleatoria pero de tal manera que estas al armarlas posean respuesta. Una vez que las piezas están creadas, se debe implementar el algoritmo genético que

permita resolver el rompecabezas. Los elementos necesarios para llevar a cabo el algoritmo genético en el tetravex según Valerio son:

- **Población inicial**: el programa creará la población inicial de forma aleatoria del tamaño del rompecabezas, cada cromosoma contendrá todas las piezas (gen) del rompecabezas sin repetirse. Cada gen está compuesto por 4 alelos.

Tamaño rompecabezas	Cantidad población
3x3	30
5x5	60
7x7	90

- **Función aptitud o fitness:** Cada población generada se debe evaluar, cuando ocurre una generación las poblaciones compiten entre sí y con sus padres.
- **Cruces:** El programa tomará la población generada y evaluada de mayor a menor para realizar los cruces y generar la cantidad de hijos indicados en la siguiente tabla para cada tipo de cruce:

Tamaño rompecabezas	Cantidad población	Cantidad de hijos por generación por tipo de cruce
3x3	30	50
5x5	60	60
7x7	90	70

  El programa aplica tres tipos diferentes de cruces, cada tipo de cruce toma la población inicial generada y la clona para realizar sus propias generaciones.
- **Mutación:** Cuando haya poblaciones con evaluación empatada, se debe aplicar un tipo de mutación que no sea aleatorio sino una alteración entre dos piezas que no le coinciden sus bordes, en el caso de mejorar la puntuación

se queda aplicada, en caso de empeoramiento en la evaluación, descartarla. La misma mutación se aplicará para las generaciones de los tres cruces diferentes.

- **Generaciones:** Cada vez que se cruza toda la población evaluada es una generación. El programa deberá realizar un ciclo para generar varias generaciones como se indica en la siguiente tabla

Tamaño rompecabezas	Cantidad población	Cantidad de hijos por generación por tipo de cruce	Cantidad de Generaciones
3x3	30	50	50
5x5	60	60	50
7x7	90	70	50

Posteriormente se realizan mediciones para determinar el rendimiento de los diferentes tipos de cruce, probando con los tres tamaños diferentes, y además se debe realizar el factor talla comparando entre dichos tamaños.

Se deben realizar pruebas sobre los diferentes tipos de cruce, con las medidas de los tres tamaños. La población inicial debe ser la misma para los tres tipos de cruce. Para calcular el factor de talla se deben realizar las siguientes comparaciones con respecto a los tamaños:

Tamaño piezas	factor
25/9	2,7
49/25	1,9
49/9	5,4

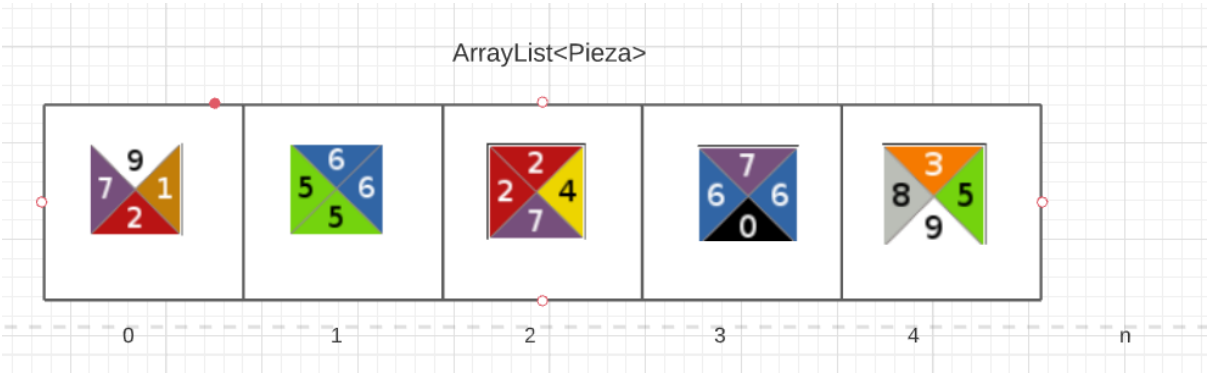
El conteo de asignaciones, comparaciones, tiempo y memoria consumida se termina de concluir con las 50 generaciones. Se debe realizar el conteo de memoria consumida (Si es un int32, se suman 32 bits; si es un varchar, se suman 8 bits). Se debe calcular el factor de crecimiento y el factor de talla de los algoritmos en base al tipo de medición O grande. De igual manera, incluir las tablas de medición análitica y empírica por cada algoritmo de cruce, siguiendo la notación O grande.

Por último, las consultas deben imprimir todas las variables de medición para cada uno de los cruces. Imprimir todos los cruces que se hicieron en la metodología genética, de la misma manera la puntuación asignada a cada cromosoma. Imprimir las mutaciones aplicadas con la puntuación asignada a cada cromosoma. Listar las cinco mejores poblaciones con su puntuación en cada cruce.

Solución del problema

- **Estructuras Utilizadas**

Globales:



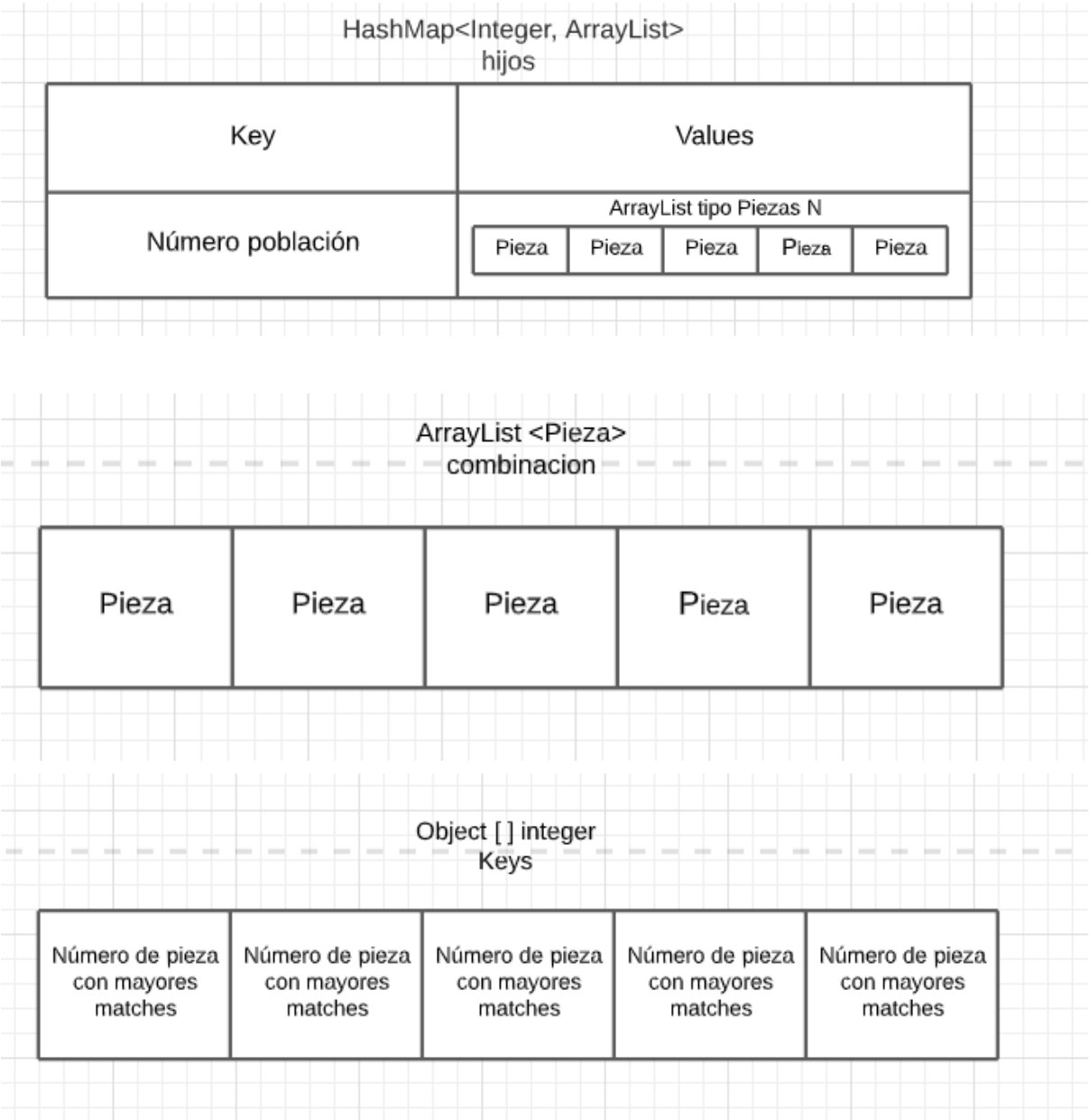
HASHMAP<Integer,Integer>
matches

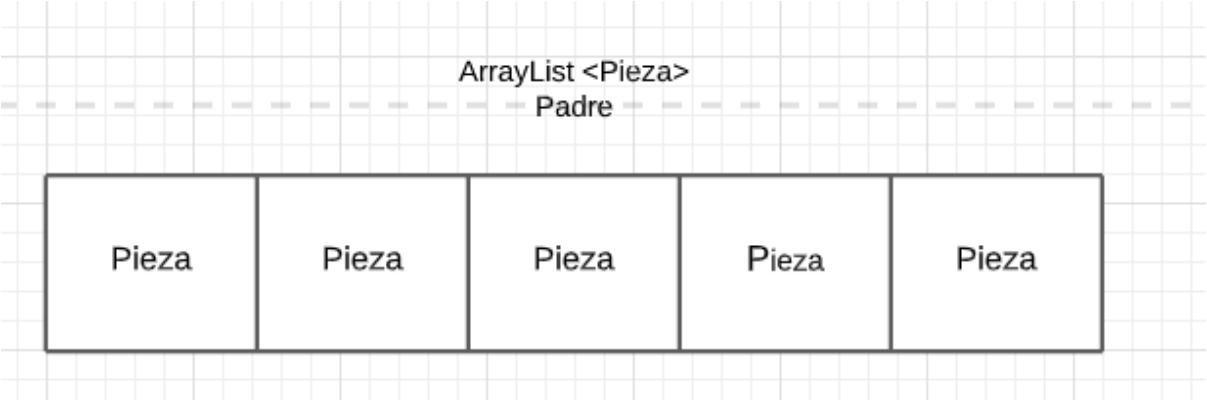
KEY	VALUE
Número de población	Puntuación de la población

HASHMAP<Integer,ArrayList>
PoblacionesIniciales

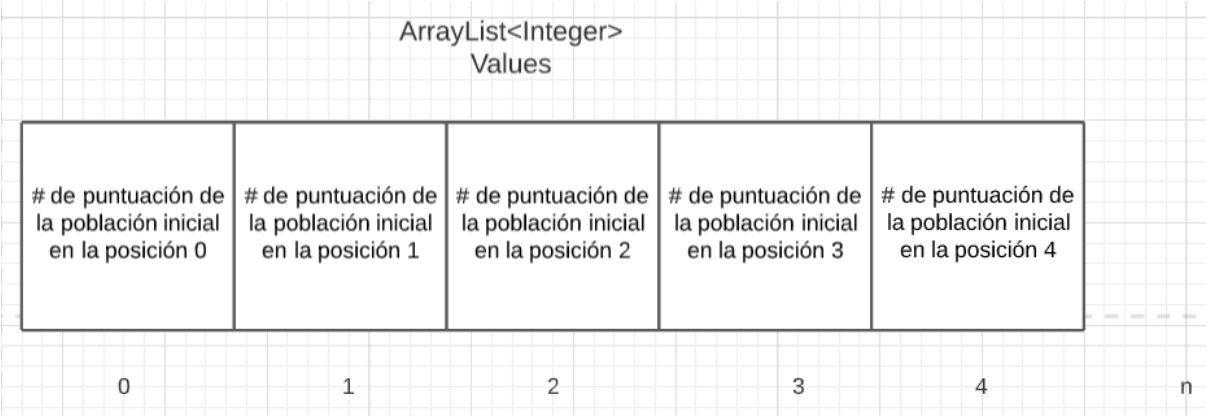
KEY	VALUE
Número de población	ArrayList de tipo Piezas N Pieza | Pieza | Pieza | Pieza | Pieza

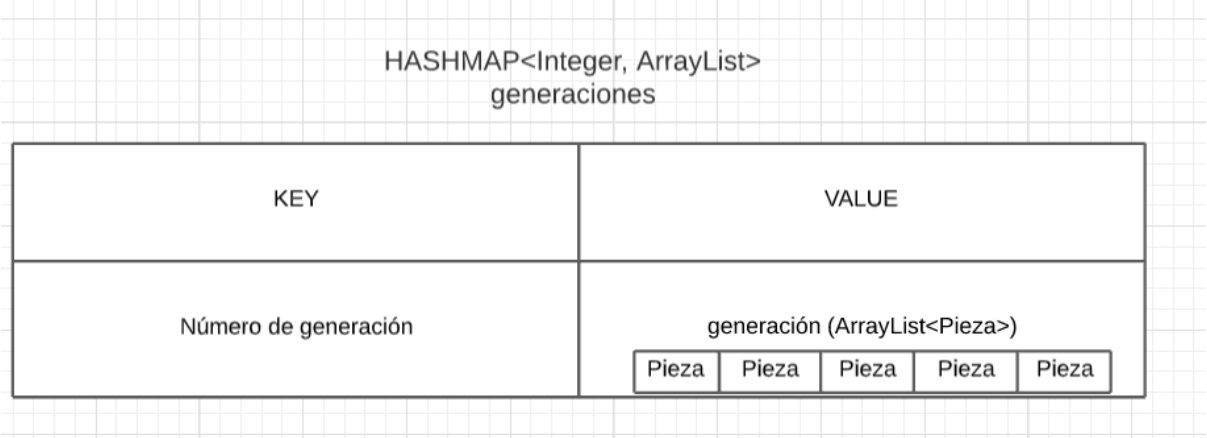
Cruce K-point:



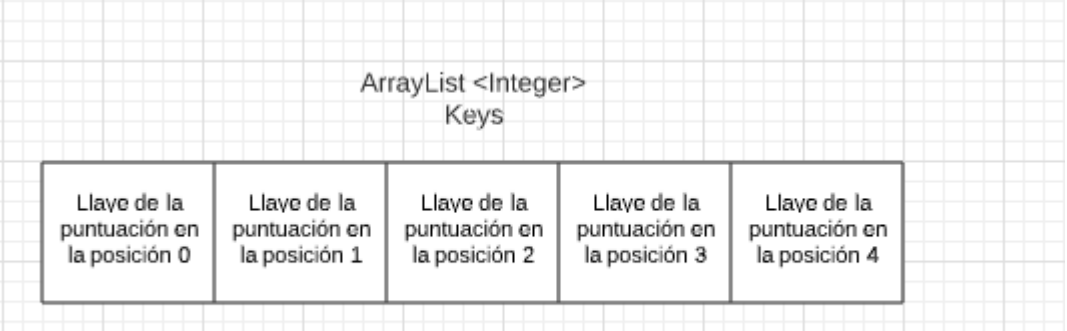


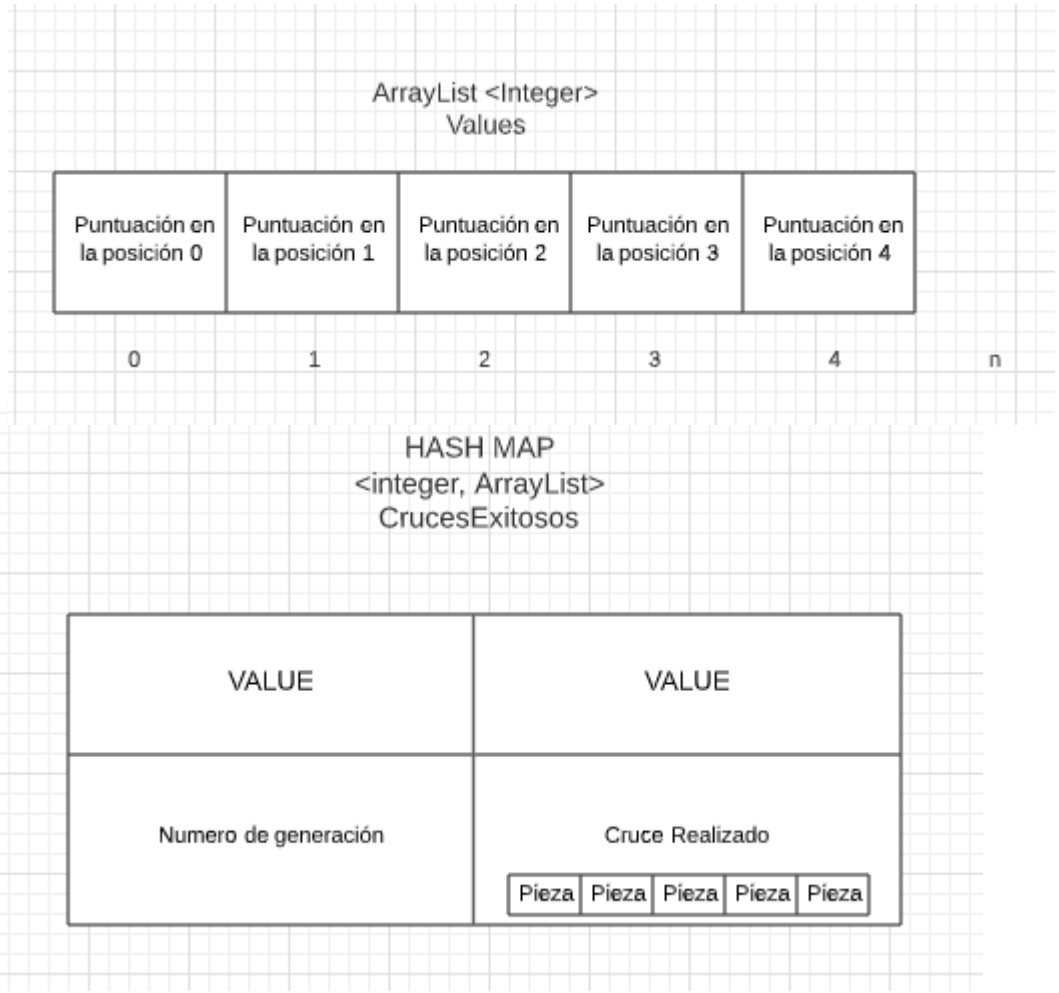
Cruce And:





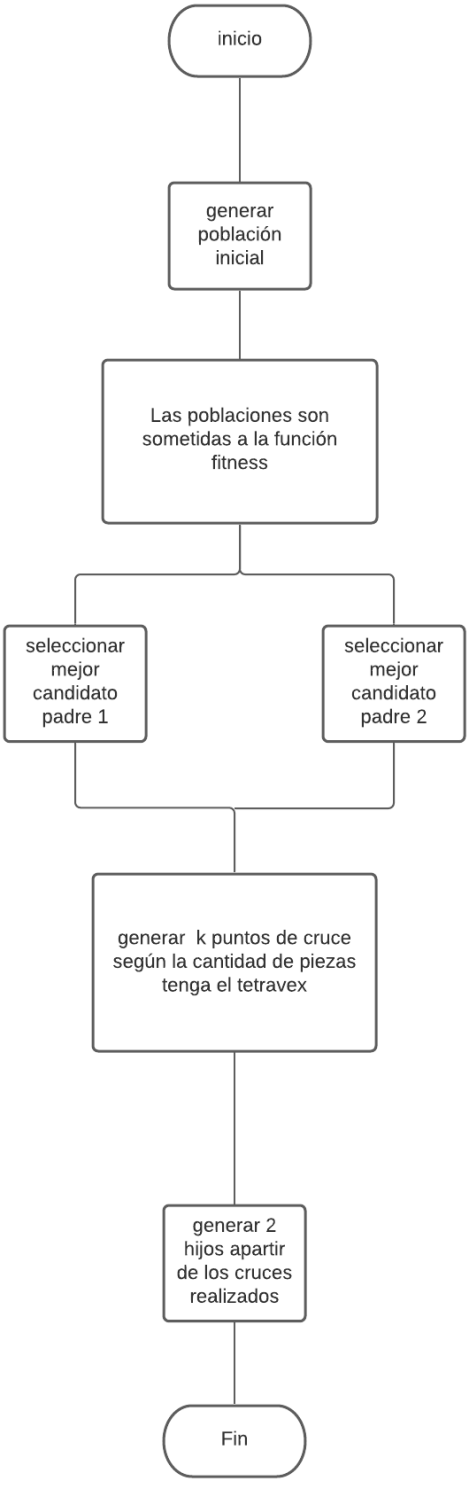
Cruce Shuffle:




ArrayList <Integer>
Values
Puntuación en la posición 0
Puntuación en la posición 1
Puntuación en la posición 2
Puntuación en la posición 3
Puntuación en la posición 4
0
1
2
3
4
n
HASH MAP
<integer, ArrayList>
CrucesExitosos
VALUE
VALUE
Numero de generación
Cruce Realizado
Pieza
Pieza
Pieza
Pieza
Pieza

- **Diagramas de flujo y descripción**

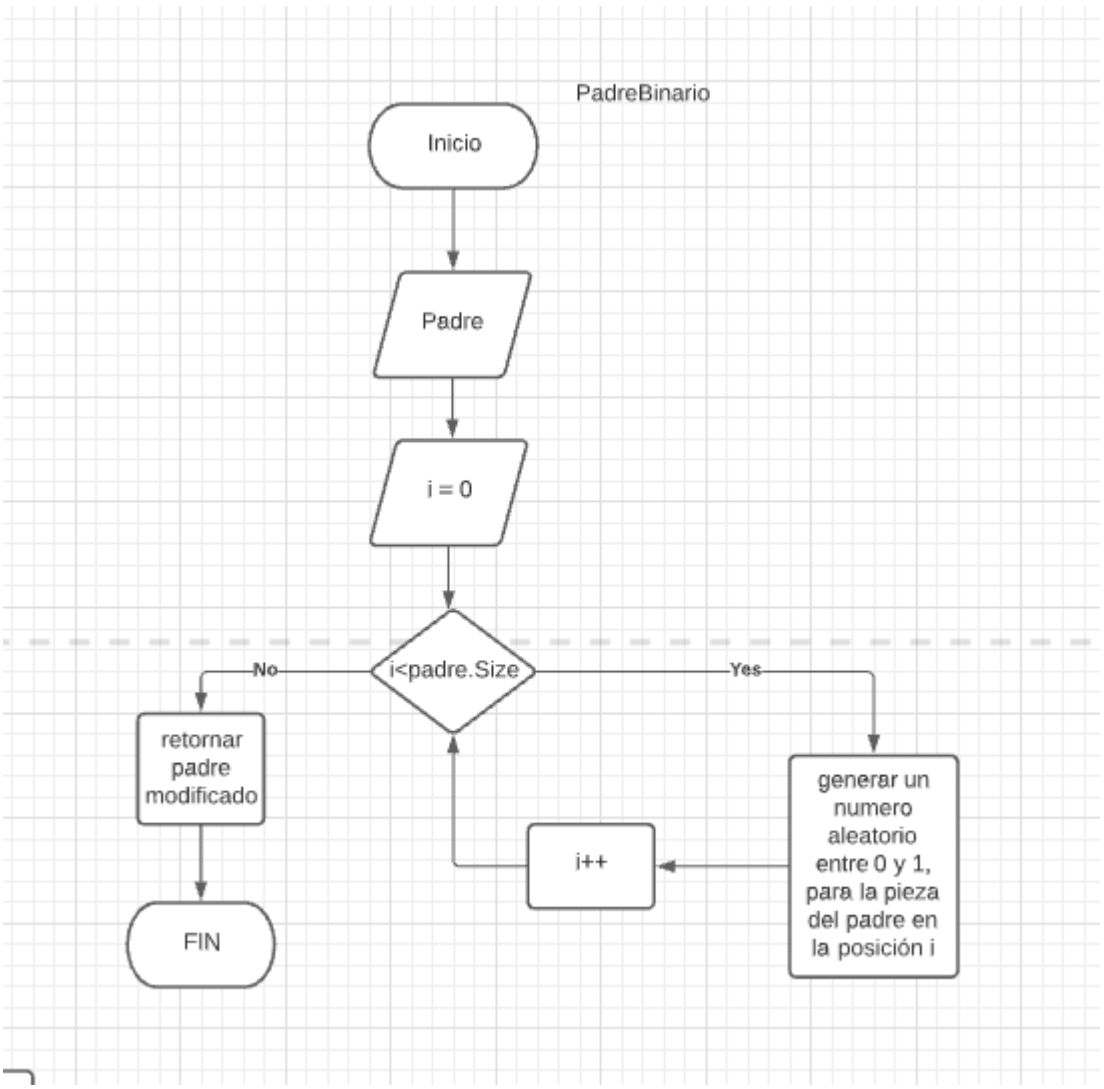
Cruce K-point.

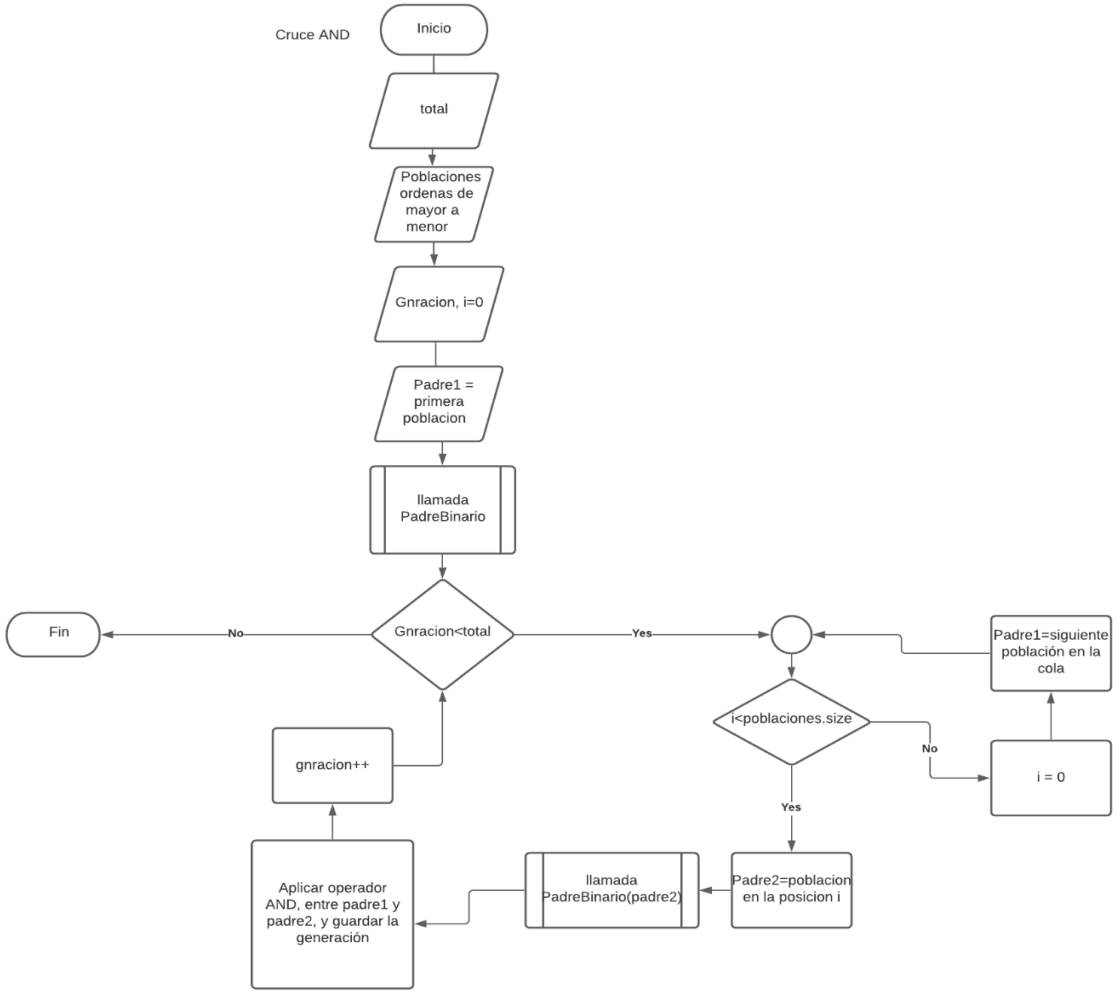


El cruce de k-puntos consiste en seleccionar a los mejores dos candidatos de la población inicial (con la myor cantidad de matches por tetravex) y se procede a generar k puntos de cruce en ambos padres dependiendo de cuantas piezas tengan los padres. En los puntos de cruce se intercambian los genes generando un hijo de ese par de padres, posteriormente se cambia de padre 2 para conseguir otro hijo , así sucesivamente hasta llegar a la cantidad de hijos pedido por la dimensión de tetravex.

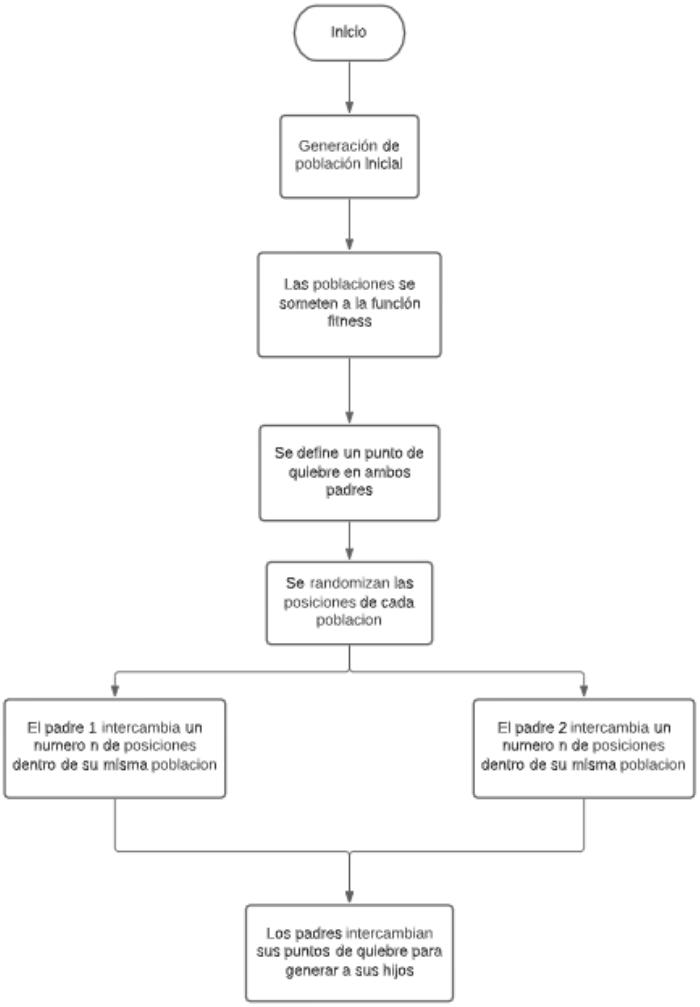
Cruce por operador lógico AND

El cruce And consiste en cruzar dos padres (tetravex) escogidos de la población inicial por medio del operador lógico AND. Una vez obtenido los progenitores, se les asigna a cada uno de sus piezas 0 o 1, una vez todos sus elementos tengan valor se procede a realizar el cruce; si una pieza de un padre tiene un 1 y la del otro tiene un 0, se escoge la que posea el 0; si ambas piezas poseen un 1, se escoge la del primer padre, si ambas poseen 0, se escoge la del segundo padre. Se sigue el procedimiento hasta haber cruzado todas las piezas de los padres. Diagrama de flujo:


PadreBinario
Inicio
Padre
i = 0
i<padre.Size
No
Yes
retornar padre modificado
FIN
generar un numero aleatorio entre 0 y 1, para la pieza del padre en la posición i
i++


Cruce AND
Inicio
total
Poblaciones ordenas de mayor a menor
Gnracion, i=0
Padre1 = primera poblacion
llamada PadreBinario
Gnracion<total
No
Fin
Yes
Padre1=siguiente población en la cola
i<poblaciones.size
No
i = 0
Yes
Padre2=poblacion en la posicion i
llamada PadreBinario(padre2)
Aplicar operador AND, entre padre1 y padre2, y guardar la generación
gnracion++

Cruce Shuffle



El algoritmo de Shuffle consiste en generar todos los posibles rompecabezas, después se someten a la función fitness para encontrar a los más aptos para el cruce y generar a los mejores hijos. Se define un punto de quiebre que después se utilizará para intercambiar las partes de los padres. Se mezclan lo máximo posible ambos padres y empiezan a intercambiar posiciones dentro de su misma población. Una vez hecho eso se realiza el cruce.

- **Función fitness**

La función se encarga de realizar la puntuación de cada población, esta recibe las poblaciones a puntuar, se recorre y por cada elemento, es decir, por cada rompecabezas se inspecciona cada pieza y se empieza a buscar otras piezas con las cuales encaja, por cada encaje que se encuentre se va sumando la puntuación. Se repite el procedimiento hasta que se hayan recorrido todas las piezas, se pasa a

la siguiente población y se termina cuando se hayan puntuado todos los tetravex pasadas por parámetro. Retorna las poblaciones puntuadas en forma de HashMap.

Análisis de resultados

Medición empírica

Nombre del algoritmo #1: _ con la combinación de 0..9

Cruce And (0-9)	***cantidad de piezas***		
Operaciones	3x3	5x5	7x7
Asignaciones	2682	8056	17738
Comparaciones	2475	7865	17473
Cantidad de líneas ejecutadas	5157	15921	35211
Tiempo de ejecución	0.2174 miliseg	10.2407 miliseg	24.85 miliseg
Memoria	95840 bits	292208 bits	647680bits
Cantidad de líneas del código	50 líneas		

Nombre del algoritmo #1: _ con la combinación de 0..9

Cruce Shuffle (0-9)	***cantidad de piezas***		
Operaciones	3x3	5x5	7x7
Asignaciones	815	2415	5335
Comparaciones	250	780	1750
Cantidad de líneas ejecutadas	1065	3195	7085
Tiempo de ejecución	1.8839 miliseg	6.5079 miliseg	12.5864miliseg
Memoria	13497 bits	34105 bits	64377 bits
Cantidad de líneas del código	38 líneas		

Nombre del algoritmo #1: _ con la combinación de 0..9

Cruce Kpoint (0-9)	***cantidad de piezas***		
Operaciones	3x3	5x5	7x7
Asignaciones	1565	4755	10581
Comparaciones	1000	4433	10288
Cantidad de líneas ejecutadas	2565	9188	20869
Tiempo de ejecución	3.5922 miliseg	6.5448. miliseg	19.5121 miliseg
Memoria	14760 bit	23432 bits	34920 bits
Cantidad de líneas del código	45 líneas		

Factor talla

Cruce AND(0-9)

Talla	Factor Talla	Factor Asig	Factor Comp	Factor cantidad de líneas ejecutadas	Memoria	Factor tiempo de ejecución
De 25/9	2,7	3	3.17	3.08	3.048	47.10
De 49/25	1,9	2.20	2.22	2.21	2.78	2.42
De 49/9	5,4	6.61	7.05	6.82	6.75	114.30

Clasificación del comportamiento de las **asignaciones**	O(N)
Clasificación del comportamiento de las **comparaciones**	O(N)
Clasificación del comportamiento de las **líneas ejecutadas**	O(N)
Clasificación del comportamiento de la **memoria**	O(N)
Clasificación del comportamiento del **tiempo de ejecución**	O(N)

Cruce Shuffle (0-9)

Talla	Factor Talla	Factor Asig	Factor Comp	Factor cantidad de líneas ejecutadas	Memoria	Factor tiempo de ejecución
De 25/9	2,7	2.96	3.12	3	2.52	3.45
De 49/25	1,9	2.20	2.24	2.21	1.88	1.93
De 49/9	5,4	6.54	7	6.65	4.76	6.68

Clasificación del comportamiento de las **asignaciones**	O(N)
Clasificación del comportamiento de las **comparaciones**	O(N)
Clasificación del comportamiento de las **líneas ejecutadas**	O(N)
Clasificación del comportamiento de la **memoria**	O(N)
Clasificación del comportamiento del **tiempo de ejecución**	O(N)

Cruce Kpoint (0-9)

Talla	**Factor Talla**	**Factor Asig**	**Factor Comp**	**Factor cantidad de líneas ejecutadas**	**Memoria**	**Factor tiempo de ejecución**
De 25/9	2,7	3.03	4.43	3.58	1.58	1.82
De 49/25	1,9	2.22	2.32	2.27	1.49	2.98
De 49/9	5,4	6.76	10.28	8.13	2.36	5.43

Clasificación del comportamiento de las **asignaciones**	O(N)
Clasificación del comportamiento de las **comparaciones**	O(N)
Clasificación del comportamiento de las **líneas ejecutadas**	O(N)
Clasificación del comportamiento de la **memoria**	O(N)
Clasificación del comportamiento del **tiempo de ejecución**	O(N)

Medición Analitica

Cruce AND

Código fuente **Solo se analiza el código del cruce**	**Medición de líneas ejecutadas en el peor de los casos (línea por línea)**
public HashMap<Integer,ArrayList> cruceAnd(HashMap<Integer,Integer> matches,HashMap<Integer,ArrayList> poblacionesIniciales,int totalG){	
ArrayList<Integer> values = new ArrayList(matches.values());	1
values.sort(Collections.reverseOrder());	1
HashMap<Integer,ArrayList> generaciones = new HashMap();	1
ArrayList<Pieza> padre1 = new ArrayList();	1
ArrayList<Pieza> padre2;	
ArrayList<Pieza> hijo;	
int puntuacionP1 = 0;	1
for(int i =0;i<matches.size();i++){	1+(n+1)+1
if(Objects.equals(matches.get(i), values.get(0))){	n
padre1 =poblacionesIniciales.get(i);	n
puntuacionP1 = matches.get(i);	n
}	
}	
int gen = 0;	1
int pob = 1;	1
int anterior = 1;	1
while(gen<totalG){	n+1
if(pob>=poblacionesIniciales.size()){	n
pob=anterior;	

for(int i =0;i<matches.size();i++){	1
if(Objects.equals(matches.get(i), values.get(pob-1))) {	(n+3)*n
padre1 =poblacionesIniciales.get(i);	n
puntuacionP1 = matches.get(i);	n
}	n
}	
}	
while(pob<poblacionesIniciales.size()){	
padre2 = poblacionesIniciales.get(pob);	n^2+1
padreBinario(padre2);	$1+n^2$
hijo = creaHijoAnd(padre1, padre2);	$n^2*(3n+4)$
generaciones.put(gen, hijo);	$(8n+5)*n^2$
gen++;	n^2
pob++;	n^2
if(gen==totalG) pob=poblacionesIniciales.size();	n^2
}	n^2
}	
return generaciones;	
}	1
public void padreBinario(ArrayList<Pieza> padre){	total: $5n^2+13n+14$
Random aleatorio = new Random();	
byte bit;	
for(int i =0;i<padre.size();i++){	1
bit = (byte) aleatorio.nextInt(2);	1
padre.get(i).setBin(bit);	n+3
	n
}	n

}
 	
public ArrayList<Pieza> creaHijoAnd(ArrayList<Pieza> padre1,ArrayList<Pieza> padre2){
ArrayList<Pieza> hijo = new ArrayList();
for(int i =0;i<padre1.size();i++){
if(padre1.get(i).getBin()==1 && padre2.get(i).getBin()==1) {
hijo.add(padre1.get(i));
}
else if(padre1.get(i).getBin()==0 && padre2.get(i).getBin()==0){
hijo.add(padre2.get(i));
}
else if(padre1.get(i).getBin()==1 && padre2.get(i).getBin()==0) {
hijo.add(padre2.get(i));
}
else {
hijo.add(padre1.get(i));
}
}
return hijo;
}
	total: 3n+4
	
	1
	n+3
	n
	n
	n
	n
	n
	n
	n
	1
	total: 8n+5

Total (la suma de todos los pasos)	$4n^2+10n+15$
Clasificación en notación O Grande	$O(N^2)$

Cruce Shuffle

Código fuente **Solo se analiza el código del cruce**	**Medición de líneas ejecutadas en el peor de los casos (línea por línea)**
public HashMap<Integer, ArrayList> ShuffleCrossover(HashMap<Integer, ArrayList> poblaciones, int cPoblaciones, int cruces,int dimension){	
HashMap<Integer, Integer> cantidadMatches = funcionFitness(poblaciones, dimension);	1
HashMap<Integer, Integer> matchesOrdenados = ordenarHashMap(cantidadMatches);	1
HashMap<Integer, ArrayList> crucesExitosos = new HashMap();	1
ArrayList<Pieza> padreUno = new ArrayList();	1
ArrayList<Pieza> padreDos = new ArrayList();	1
ArrayList<Pieza> hijoUno = new ArrayList();	1
ArrayList<Pieza> hijoDos = new ArrayList();	1
ArrayList<Pieza> hijoUnoClone = new ArrayList();	1
ArrayList<Pieza> hijoDosClone = new ArrayList();	1
ArrayList<Integer> values = new ArrayList(matchesOrdenados.values());	1
int contadorCruces = 0;	1
int contadorPoblaciones = 1;	1
values.sort(Collections.reverseOrder());	1
Object[] keys = matchesOrdenados.keySet().toArray();	1
padreUno = poblaciones.get(keys[0]);	1
Collections.shuffle(padreUno);	1
while (contadorCruces < cruces){	1
int breakPoint = (int) (Math.random() * (dimension*dimension + 1));	n + 1

if(contadorPoblaciones == cPoblaciones){	n
padreUno = poblaciones.get(keys[1]); e	n
contadorPoblaciones = 3;	n
}	n
padreDos = poblaciones.get(keys[contadorPoblaciones]);	n
Collections.shuffle(padreDos);	n
for (int i = 0; i < breakPoint; i++) {	n^2+3
hijoUno.add(padreUno.get(i));	n^2
hijoDos.add(padreDos.get(i));	n^2
}	
for (int i = breakPoint; i < dimension*dimension; i++) {	
hijoUno.add(padreDos.get(i));	n^2+3
hijoDos.add(padreUno.get(i));	n^2
}	n^2
hijoUnoClone = (ArrayList<Pieza>) hijoUno.clone();	
hijoDosClone = (ArrayList<Pieza>) hijoDos.clone();	n
crucesExitosos.put(contadorCruces, hijoUnoClone);	n
contadorCruces ++;	n
crucesExitosos.put(contadorCruces, hijoDosClone);s	n
contadorCruces++;	n
hijoUno.clear();	n
hijoDos.clear();	n
contadorPoblaciones++;	n
}	
return crucesExitosos;	
}	1

Total (la suma de todos los pasos)	$6n^2+16n+25$
Clasificación en notación O Grande	$O(N^2)$

Cruce Kpoint

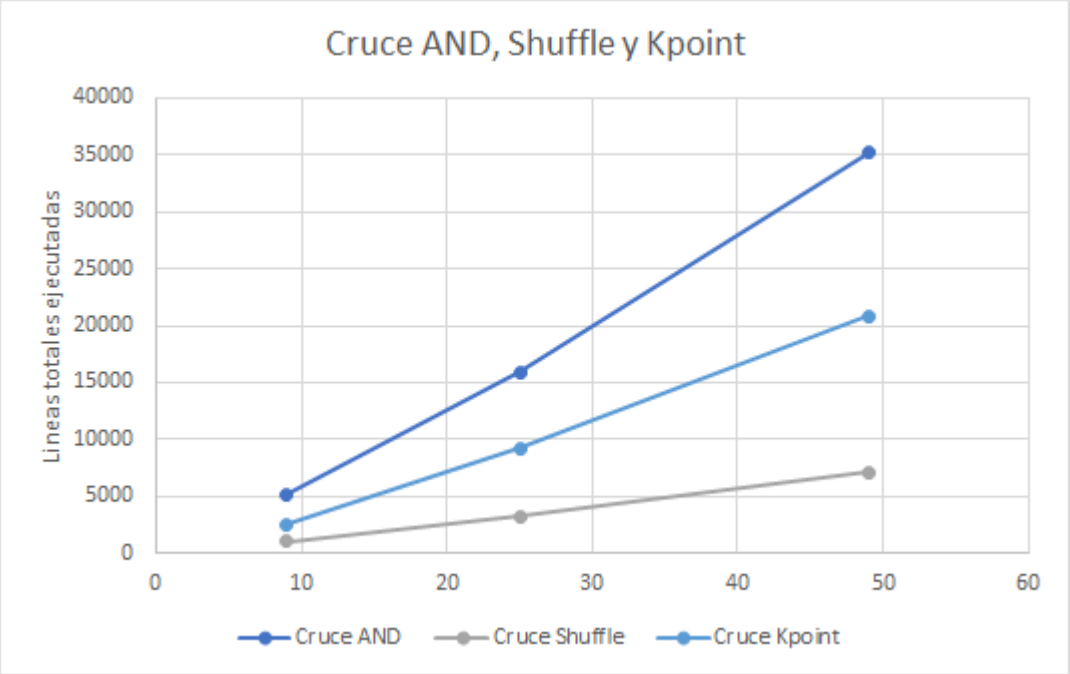
Código fuente **Solo se analiza el código del cruce**	**Medición de líneas ejecutadas en el peor de los casos (línea por línea)**
public HashMap<Integer, ArrayList> kPoint (HashMap<Integer,Integer> cantidadMatches, HashMap<Integer, ArrayList> poblacion, int hijosP, int limite){	
HashMap<Integer, ArrayList> hijos = new HashMap(); ArrayList<Pieza> combinacion = new ArrayList();	1
Object[] keys = cantidadMatches.keySet().toArray();	
Random cantidadCruces = new Random();	1
int numero = cantidadCruces.nextInt(limite+1)+2;	1
int cantidadSeparacion = (limite*limite)/numero;	1
boolean cambio= true;	1
int cantiC =0;	1
int contaP1 = 0;	1
int keyHijo= 0;	1
int contaP2 =1;	1
int aux =1;	1
ArrayList<Pieza> padre1 = poblacion.get(keys[contaP1]);	
ArrayList<Pieza> padre2 = poblacion.get(keys[contaP2]);	1 --------13

while (hijosP > keyHijo){	n+1
for (int m=0 ; m<(limite*limite); m++){	(n+3)n = 3n+n^2
if((cambio == true) && (cantiC < (cantidadSeparacion-1))){	(n+2)n = 2n+n^2
combinacion.add(padre1.get(m));	n*n = n^2
cantiC++;	n*n=n^2
}	
else if ((cambio == false) && (cantiC < (cantidadSeparacion-1))){	(n+2)n = 2n+n^2
combinacion.add(padre2.get(m));	n*n= n^2
cantiC++;	n*n=n^2
}	
else{	
if((cambio == true)){combinacion.add(padre1.get(m));}	(n+1)*n=n+n^2
else if ((cambio == false)){combinacion.add(padre2.get(m));}	(n+1)*n= n+n^2
cantiC=0;	n*n=n^2
cambio= !cambio;	n*n=n^2
}	
}	
contaP2++;	n
if(contaP2< poblacion.size()){	n+1
padre2 =poblacion.get(keys[contaP2]);	n
}	

if (contaP2 >= poblacion.size()){
contaP1++;
padre1 = poblacion.get(keys[contaP1]);
aux++;
contaP2 = aux;
padre2 = poblacion.get(keys[contaP2]);
}
 	
hijos.put(keyHijo, combinacion);
keyHijo++;
combinacion = new ArrayList();
}
return hijos;
 	
}
	n+1
	n
	n
	n
	n
	n
	
	n
	n
	n
	
	1

Total (la suma de todos los pasos)	$12n^2+20n+17$
Clasificación en notación O Grande	$O(N^2)$

Medición Gráfica



Resultados Finales

Requerimientos del Algoritmo	Estado	¿Qué se puede mejorar?
Población Inicial	completo y funcional	El programa crea la población inicial de forma correcta, con un tamaño de rompecabezas elegido por el usuario, con piezas irrepetibles y cada pieza posee 4 números. Cada rompecabezas posee una solución.
Función Aptitud o Fitness	completo y funcional	La función aptitud o fitness funciona de forma correcta, retornando la puntuación de cada rompecabezas para poder identificar cuales son las mejores opción para realizar los cruces correspondientes.
Cruce AND	completo y funcional	El cruce funciona correctamente, se podría mejorar la eficiencia, puesto el bucle de asignar padre1 se podría aplicar de otra manera más reducida.
Cruce Shuffle	completo y funcional	El cruce funciona de manera correcta, ya que primero escoge los dos padres, luego los mezcla, y después se decide un punto de quiebre para realizar el cruce.
Cruce KPoint	completo y funcional	Funciona de manera correcta al tomar los dos padres y combinarlos además de que genera la cantidad pedida de hijos por población. Se podría la manera de cambiar de padres para generar otro hijo ya que usa muchos contadores para poder llevar esto acabo
Mutación	Incompleto	No se logró implementar el proceso de mutación por falta de tiempo para crear un método funcional. Se puede mejorar la distribución del tiempo por parte de los integrantes.
Generaciones	completo y funcional	Todas las generaciones de los cruces se imprimen de forma correcta, imprimiendo la cantidad de cruces elegidos por el usuario, seguido de los mejores cinco hijos en cuanto a puntaje.

Conclusiones

Después de la elaboración y análisis tanto empirico como análitico de los cruces se concluye lo siguiente:

- Respecto a cantidad de asignaciones y comparaciones, es decir, la cantidad de instrucciones, el cruce Shuffle es el más óptimo.
- En tiempo de ejecución, el cruce Shuffle es el que menor tiempo obtuvo para las tres dimensiones.
- En consumo memoria, el cruce que más optimizo el uso de esta fue el cruce K-Point
- En líneas de código, el cruce Shuffle tiene menos que el resto.
- En análisis análitico, todos los cruces son de comportamiento de cuadrático, pero el que consiguió una breve mejora respecto a los demás es el cruce por operador lógico And.

Según los puntos anteriores, en promedio, el cruce que mejores resultados obtuvo es el cruce Shuffle.

Los tres algoritmos tienen un grado de incertidumbre bastante alto, ya que poseen varias partes donde la aleatoriedad es parte importante de la resolución, aun así el el cruce de Shuffle curiosamente se vuelve más efectivo conforme la talla crece a comparación de los otros dos algoritmos, un ejemplo de esto es el tiempo de ejecución del cruce AND en un tetravex de 7x7 que corresponde 24.85 milisegundos el doble del cruce Shuffle que corresponde a 12.5864 milisegundos.

Recomendaciones

- Se recomienda el uso de HashMaps como diccionarios para guardar la información de las poblaciones.
- Usar la modularización para realizar funciones que ayuden en el proceso de validar requisitos para llegar la solución, por ejemplo funciones para saber si se sigue en una fila o si se debe cambiar, en el rompecabezas.
- Se recomienda usar la biblioteca Collections para mezclar las piezas de los padres antes del cruce Shuffle.
- Se aconseja investigar las diferentes estructuras de datos (HashMaps, ArrayList, LinkedList) y sus funcionamientos, esto permite que el desarrollador use al máximo todo el provecho de cierta estructura y así dé como resultado un programa eficiente y efectivo.
- Por parte de los programadores de este trabajo se incentiva al lector al estudio de algoritmos genéticos hasta poder llegar al conocimiento necesario para ser usados de manera aplicativa y eficientes en áreas como: IA, búsqueda y optimización.

Referencias

Arranz, J., & Parra, A. (2021). *Algoritmos Genéticos*. uc3m. http://www.it.uc3m.es/jvillena/irc/practicas/06-07/05.pdf

Get Keys From HashMap in Java. (2020, 10 diciembre). DelftStack. https://www.delftstack.com/howto/java/how-to-get-keys-from-hashmap-in-java/

How to Sort a HashMap by Value in Java? (2021). JournalDev. https://www.journaldev.com/43792/sort-hashmap-by-value-java

Introduction To Soft Computing - IITKGP. (2018, 24 febrero). *Crossover techniques* [Vídeo]. YouTube. https://www.youtube.com/watch?v=ZhfXXf7bnYE&ab_channel=IntroductionToSoftComputing-IITKGP

Umbarkar, A. J., & Sheth, P. D. (2015). *Crossover Operators In Genetic Algorithms*. IJSC. http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf

Valerio Solís, L. (2021, 13 octubre). *Algoritmos Genéticos* [Diapositivas].PowerPoint.https://tecdigital.tec.ac.cr/dotlrn/classes/CA/IC3002/S-2-2021.SC.IC3002.50/file-storage/#/114282706

Valerio, L. (2021). *Medición de algoritmo genético*. TecDigital. https://tecdigital.tec.ac.cr/dotlrn/classes/CA/IC3002/S-2-2021.SC.IC3002.50/file-storage/view/Proyectos%2FSegundo_proyectos_IIS_2021.pdf

Bitácora

Actividad o tarea	Integrante	Fecha	Hora invertidas	Observaciones/ Pendientes
Elaboración de la introducción y minuta	Melissa Alguera Castillo	1/11/2021	1.5	La introducción fue revisada por los demás integrantes del grupo y aceptada.
Elaboración del análisis del problema	Richard León Chinchilla	1/11/2021	1.5	El análisis fue revisado por los demás integrantes del grupo y aceptado.
Elaboración del análisis del problema	Adrián Herrera Segura	1/11/2021	0.5	El análisis fue revisado por los demás integrantes del grupo y aceptado.
Diseño de generación de piezas	Melissa Alguera	2/11/2021	1.5	Funcionamiento del generador de piezas. Pendiente creación Algoritmo genético.
Diseño de generación de piezas	Richard León Chinchilla	2/11/2021	1.5	Funcionamiento del generador de piezas. Pendiente creación Algoritmo genético.
Diseño y creación de generación de piezas	Adrian Herrera Segura	2/11/2021	1.5	Funcionamiento del generador de piezas. Pendiente creación Algoritmo genético.

Creación función fitness (programadora)	Melissa Alguera Castillo	10/11/2021	2.5	Pendiente algoritmos de cruces y faltan pruebas de funcionamiento del fitness.
Creación función fitness	Richard León Chinchilla	10/11/2021	2.5	Pendiente algoritmos de cruces y faltan pruebas de funcionamiento del fitness.
Creación función fitness	Adrian Herrera Segura	10/11/2021	2.5	Pendiente algoritmos de cruces y faltan pruebas de funcionamiento del fitness.
Diseño y diagrama de flujo de cruce K-point	Melissa Alguera Castillo	11/11/2021	2	Pendiente la programación del cruce
Diseño y diagrama de flujo de cruce por operador AND	Richar León Chinchilla	11/11/2021	2	Pendiente la programación del cruce
Investigación de cruce a usar	Adrian Herrera Segura	11/11/2021	0.5	Pendiente diseño,diagrama, y programación del cruce
Construcción cruce K-point	Melissa Alguera Castillo	13/11/2021	1.5	Pendiente terminar el cruce K-point
Construcción cruce AND	Richard León Chinchilla	13/11/2021	1.5	Pendiente terminar el cruce AND
Construcción cruce Shuffle	Adrian Herrera Segura	13/11/2021	1.5	Pendiente terminar el cruce Shuffle
Construcción cruce K-point	Melissa Alguera Castillo	14/11/2021	3	Pendiente finalizar cruce K-point

Construcción cruce AND	Richard León Chinchilla	14/11/2021	3	Cruce terminado
Construcción cruce Shuffle	Adrián Herrera Segura	14/11/2021	3	Cruce terminado, pendiente método de impresión
Finalización de Cruce Kpoint	Melissa Alguera Castillo	15/11/2021	2.5	Pendiente impresión top 5 generaciones
Finalización Cruce And	Richard León Chinchilla	15/11/2021	2.5	Pendiente impresión top 5 generaciones
Finalización cruce Shuffle	Adrián Herrera Segura	15/11/2021	2.5	Pendiente impresión top 5 generaciones
Impresiones de cruce Kpoint	Melissa Alguera Castillo	16/11/2021	6	Impresiones listas, mediciones empiricas, falta medicion analitica, factor talla, clasificacion
Impresiones cruce AND	Richard León Chinchilla	16/11/2021	6	Impresiones listas, mediciones empiricas, falta medicion analitica, factor talla, clasificacion
Impresiones cruce Shuffle	Adrián Herrera Segura	16/11/2021	5	Impresiones listas, mediciones empiricas, falta medicion analitica, factor talla, clasificacion
Terminar la documentación externa	Melissa Alguera Castillo	17/11/2021	6	
Terminar la documentación externa	Richard León Chinchilla	17/11/2021	6	

Terminar la documentación externa	Adrián Herrera Segura	17/11/2021	6	

Minuta

Fecha-Hora	Medio	Participantes	Asuntos a Tratar	Acuerdos	Asuntos Pendientes
Viernes 29 octubre 9am-12pm	Discord	Adrian Herrera Richard León Paola Alguera	Tarea de instigación bibliografía para comenzar el proyecto	Dia de inicio del proyecto	Empezar el trabajo escrito.
Lunes 1 noviembre 8am- 11am	Discord	Adrian Herrera Richard León Paola Alguera	Redacción de la introducción y análisis del problema	Pensar un diseño para el algoritmo individualmente.	Continuar con la redacción del documento escrito y la inicialización de la parte programada
Martes 2 noviembre 8:30 am-10:40am	Discord	Adrian Herrera Richard León Paola Alguera	Continuar con la redacción del documento escrito y la inicialización de la parte programada	Pensar maneras de elaborar la función fitness	Construcción de la función fitness
Miércoles 10 noviembre 8:30 am-10:40am	Discord	Adrian Herrera Richard León Paola Alguera	Construcción de la función fitness	Pensar como elaborar los algoritmos de cruces	construcción de las funciones de cruces
Jueves 11 noviembre 8:30 - 11:00 am	Discord	Adrian Herrera Richard León Paola Alguera	Elección de algoritmos de cruce a implementar	Lógica aplicativa al programa	Iniciar con la programación de estos algoritmos
Sábado 13 de noviembre 8:30 - 11:00	Discord	Adrian Herrera Richard León Paola Alguera	Construcción cruces	Cada miembro desarrolla un cruce	Inició con la construcción de los cruces.
Domingo 14 de	Discord	Adrian Herrera Richard León	Construcción cruces	Cada miembro	Desarrollo y

noviembre 2:00 - 6:40 p.m		Paola Alguera		desarrolla un cruce	finalización de los cruces.
Lunes 15 9:00 - 12:00	Discord	Adrian Herrera Richard León Paola Alguera	Construcción cruces	Cada miembro desarrolla un cruce	Impresión cruces y conteo de asig y comp
Martes 16 8:00 - 4:30	Discord	Adrian Herrera Richard León Paola Alguera	Impresión de cruces y mediciones	Cada miembro termina su cruce	Terminar tablas de mediciones
Miércoles 17 8:30- 4:00 p.m	Discord	Adrian Herrera Richard León Paola Alguera	Finalización del proyecto		